Foundations of Computer Security
Lecture 27: Storing the ACM

Dr. Bill Young
Department of Computer Sciences
University of Texas at Austin
Recall our earlier claim: Any access control policy can be represented by an *access control matrix* (ACM).

<table>
<thead>
<tr>
<th></th>
<th>object\textsubscript{1}</th>
<th>\ldots</th>
<th>object\textsubscript{k}</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject\textsubscript{1}</td>
<td>$A_i, A_j$</td>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>\ldots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subject\textsubscript{n}</td>
<td>$A_l$</td>
<td></td>
<td>$A_i, A_m$</td>
</tr>
</tbody>
</table>

The ACM gives an explicit representation of every access permitted by every subject to every object.
You could build an explicit ACM for any access control system (e.g., BLP, Biba, RBAC, etc). But we usually don’t. Why not?

Three common alternatives exist:

1. Maintain a set of rules to compute access permissions “on the fly” based on attributes of subjects and objects.
2. Store the permissions with objects. This is called an access control list (ACL).
3. Store the permissions with subjects. This is called a capability-based system.
An access control list (ACL) stores permissions with the objects of the system.

It contains pairs of the form \(< S, P >\), listing the set of permissions \(P\) that subject \(S\) currently holds to the object.

Any request by subject \(S\) for access \(A\) to object \(O\), means checking whether \(A \in P\) for the pair \(< S, P >\) on \(O\)'s access control list.

Unix/Linux, Mac OS, and Windows all store permissions by ACL.

| Subject \(j\) | Object 
|----------------|----------------|
| Subject \(1\) | RW 
| Subject \(2\) | R 
| ... | ... 
| Subject \(n\) | X 

Unix/Linux, Mac OS, and Windows all store permissions by ACL.
Some systems store permissions with subjects rather than objects. These are called *capabilities*.

<table>
<thead>
<tr>
<th>Subject $n$</th>
<th>Obj 1</th>
<th>Obj 2</th>
<th>...</th>
<th>Obj $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>RW</td>
<td></td>
<td>W</td>
</tr>
</tbody>
</table>

Each subject $S$ maintains a collection of pairs $<O, A>$, meaning that $S$ has current permission to perform access $A$ to object $O$. To obtain access, the subject must present an appropriate capability. Thus a capability is a type of “ticket.”

Many capability based systems also permit passing capabilities from one subject to another, under controlled circumstances.
Possession of a capability is *de facto* evidence of permission. Therefore, no access check is required. But to maintain security, it is necessary to ensure that capabilities can’t be *forged* or *altered*.

Historically, various approaches have been used to protect the integrity of capabilities:

- Extend each memory location with an additional bit indicated whether or not the location contains a capability; only the OS can manipulate capabilities.
- Store capabilities in specially protected memory.
Any access control system can be represented by an access control matrix.

Storing the matrix explicitly is expensive and usually unnecessary.

Access information is often stored: implicitly as a series of rules, with each object as an access control list, or with each subject as a collection of capabilities.

**Next lecture:** Information Theory